On the Understandability of Coupling-Related Practices in Infrastructure-as-Code Based Deployments

Pierre-Jean Quéval[a,b], Nicole Elisabeth Hörner[a,b], Evangelos Ntentos[a], Uwe Zdun[a]

[a] *University of Vienna, Faculty of Computer Science, Research Group Software Architecture, Vienna, Austria*
[b] *University of Vienna, Doctoral School Computer Science, Vienna, Austria*

Abstract

Infrastructure as Code (IaC) empowers software developers and operations teams to automate the deployment and management of IT infrastructure through code. This is particularly valuable for continuously released deployments such as microservices and cloud-based systems. IaC technologies offer flexibility in provisioning and deploying application architectures. However, if the structure is not well-designed, it can lead to severe issues related to coupling aspects. Unfortunately, a lack of comprehensive coupling guidelines for IaC makes ensuring adherence to best practices challenging. Leveraging IaC-based models, metrics, and source code can enhance the comprehension and implementation of coupling measures.

Our objective was to investigate how developers understand information derived from system source code and compare it to formal IaC system diagrams and metrics. We conducted a controlled experiment involving a group of participants to evaluate the understandability of IaC system architecture descriptions through source code inspection and formal representations.

We hypothesized that providing formal IaC system diagrams and metrics as supplementary materials would improve the understanding of IaC coupling-related practices measured by task *correctness*. We also expected that these supplementary resources would lead to a significant increase in task *duration* and that there would be a notable correlation between *correctness* and *duration*.

The results suggest that including formal IaC system diagrams and metrics as supplementary materials significantly enhances the comprehension of IaC coupling-related practices, as indicated by task *correctness*. Moreover,

providing these formal representations does not significantly prolong task *duration*, indicating that they do not hinder understanding. A substantial correlation between task *correctness* and *duration* is evident when formal IaC system diagrams and metrics are available.

Keywords: Infrastructure as code, modeling, best practices, controlled experiment, empirical software engineering, empirical study

1. Introduction

Today, numerous microservice-based systems undergo rapid releases, leading to frequent alterations not only in the system's implementation but also in its infrastructure and deployment processes [1, 2]. Additionally, there is a substantial rise in the number of infrastructure nodes required by these systems [2], and how these elements are managed and structured can considerably influence development and deployment procedures. IaC offers the capability to automate the provisioning and management of infrastructure nodes through reusable scripts, as opposed to manual operations [3]. IaC ensures that a provisioned environment remains consistent each time it's deployed in the same configuration, with configuration files housing infrastructure specifications, simplifying the process of editing and distributing configurations [3, 4]. Furthermore, IaC contributes to enhancing consistency and promoting loose coupling by segregating deployment artifacts based on the responsibilities of services and teams.

Infrastructure stacks can be employed to organize the deployment infrastructure effectively. An infrastructure stack represents a collection of infrastructure elements and resources defined, provisioned, and updated as a cohesive unit [3]. An incorrect structural approach can result in significant issues if coupling-related considerations are overlooked.

1.1. Problem Statement

Despite its advantages, the flexibility of IaC can lead to coupling issues, which can impact the maintainability and scalability of systems. Poorly organized coupling can result in components being tightly interdependent. This makes it difficult for the system to grow, modify, or scale. While some research addresses coupling at the code level in IaC [5, 6], few studies focus on coupling at the design or architecture level. Architectural-level coupling

issues, such as tightly integrated services or components, limit the ability to independently deploy, update, or scale individual parts.

Addressing coupling at the design level is essential for large-scale IaC deployments, where multiple components must interact while remaining decoupled for independent development and deployment. Unfortunately, formal guidelines and documented best practices for managing coupling in IaC remain limited [7, 8]. Existing industry guidelines [9, 10] often provide general recommendations but lack specificity for ensuring loose coupling in complex IaC architectures.

Consider a typical microservices setup using Infrastructure as Code (IaC). The level of connection between services, infrastructure parts, and configurations plays a big role in how flexible the system is. For example, if multiple microservices share the same environment, changing one service can impact the others, making it harder to scale or modify services independently. To assess the coupling in an IaC script, it's not enough to look at individual or local changes; you also need to consider how services are linked, how they depend on infrastructure and the entire deployment configuration.

It has been customary to use diagrams, particularly UML models, to help practitioners better understand software designs and infrastructures. However, contrary to what could be expected, those haven't consistently been shown to improve understandability or were even shown to be detrimental [11, 12, 13]. For this reason, advocating the use of a specific formalism in a specific use case should be well-supported by empirical evidence.

1.2. Research Objectives, Hypotheses, and Results

Our work includes two primary hypotheses, which we aim to explore:

H1 *We hypothesize that semi-formal deployment architecture models (based on a formal modeling method), specifically designed to abstract over IaC technologies, can improve the understanding of coupling-related aspects in IaC-based deployment architectures. These models offer a clearer big-picture view compared to continuous manual inspection of IaC scripts.* For example, when analyzing the dependencies between different components and modules in an IaC system, the interactions and coupling between resources (such as shared variables or resources across modules) must be examined. A model provides an abstraction over multiple IaC technologies and their coupling structures, helping developers visualize the connections between components and resources intuitively.

H2 *We hypothesize that formally defined metrics, precisely measuring the extent of coupling-related practices in IaC architectures, can facilitate a more effective assessment of these practices and, in turn, enhance the comprehensibility of the deployment architecture.*
For instance, if metrics automatically measure the degree of coupling between modules or components before and after a commit, it could signal whether the architecture adheres to loose coupling practices. This would eliminate the need to manually inspect the entire architecture or IaC source code to assess coupling-related practices.

We focus on an empirical evaluation of how formal models and metrics impact the comprehension of coupling-related practices in IaC. Rather than grounding our hypotheses in cognitive theories, we aim to measure practical effects in a software engineering context. Earlier works suggest this relationship[14, 15].

While cognitive theories have been used to analyze understandability, our study does not aim to validate or refute such theories. Instead, we focus on the empirical assessment of structured representations in practical IaC deployment tasks.

To support developers in working with models and metrics, as outlined in our hypotheses, our study introduces (see Section 2.3):

1. a structured modeling approach that extends UML-based, semi-formal deployment architecture diagrams by incorporating details that represent decision options commonly found in IaC scripts. This method abstracts away variations between different IaC technologies studied in our previous research [16];
2. a formally defined metric for each decision option [16], enabling the automatic assessment of its presence in a given deployment architecture.

We propose that using system models and metrics can improve how developers understand IaC security practices, as reflected in task *correctness* and *duration*. We use a more detailed UML notation that shows more details than possible in a deployment diagram and also models the instance level. Our findings show that participants who had access to a system source code repository along with metrics and semi-formal deployment architecture models demonstrated a significantly better grasp of IaC coupling practices compared to those who only worked with the source code repository. The additional resources noticeably enhanced their understanding without compromising efficiency. Overall, this work helps bridge the gap in empirical

research on architectural security concerns specific to IaC and deployment architectures.

We hypothesize that measuring understandability through the dependent variables' correctness and duration would reveal a significantly better understanding of IaC coupling practices when models of a given system are provided. More specifically, correctness is used to measure the effectiveness and duration is used to measure the efficiency of understanding IaC coupling practices

To test these hypotheses, we focused on two formally modeled ADDs frequently applied in IaC systems. These decisions include 4 design-level best practices related to coupling, derived from a detailed study of the gray literature and 9 open-source IaC-based systems developed by practitioners. Our investigation is conducted using two open-source systems. The relevant background is discussed in Section 2.

1.3. Structure of this Article

We provide an overview of coupling in IaC-based deployments, metrics, and related studies in Section 2. Next, we describe the planning of this study in Section 3. In Section 4, we detail the execution of the experiment, and the results are presented in Section 5. We discuss the results in Section 6. Section 7 discusses related work before concluding the article in Section 8.

2. Background

In this section, we summarize the background of this work, namely architectural coupling-related decisions for IaC, and models and metrics for IaC.

2.1. Infrastructure Stacks, Execution Environment, and Infrastructure Resources

Central concepts for coupling related aspects of IaC are infrastructure stacks, execution environments, and infrastructure resources. We explain these partly overlapping terms and how they are related in this section.

Infrastructure Stacks can organize the deployment infrastructure, which refers to the set of hardware, software, and networking resources required to deploy and run applications, services, and systems in a production environment [3]. The term *Infrastructure Resources* is rather generic, referring to any kind of infrastructure elements such as execution environments (such as

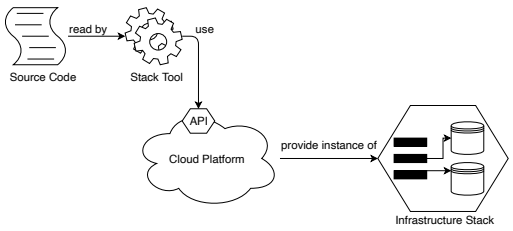

Figure 1: The lifecycle of an infrastructure stack

Containers, Virtual Machines (VMs), or Cloud Environments), servers (e.g., database servers or Web Servers), or devices (such a Local or Cloud Servers or Clusters).

The term *Execution Environment* is used here to denote the virtual subset of Infrastructure Resources, meaning a service or database does not directly run on a device but in a software environment. Examples of execution environments are Virtual Machines, Containers, or Cloud Environments such as AWS Lambda. Please note that execution environments can be nested. For instance, a VM can be part of a Production Environment, which runs on a Public Cloud Environment. Execution environments run on Devices (e.g., a local Server or a Cloud Server).

Figure 1 shows the lifecycle of an infrastructure stack. The resources and services that an infrastructure platform offers are the elements of a stack, and they are described by source code. For instance, a stack might consist of computing resources (e.g., a virtual machine), storage resources (e.g., disk volume), and network resources (e.g., a subnet). A stack management tool reads the source code for the stack and assembles the defined elements in the code to provision an instance of the infrastructure stack using a cloud platform's API.

2.2. Coupling in Infrastructure as Code Deployments

Naturally, the infrastructure of a software system is a highly critical aspect of the system; thus, infrastructure automation such as IaC needs to operate in a loosely coupled fashion. This section explains two categories of the most common coupling-related best practices or patterns for IaC architectures. In the hands-on task, you must inspect the IaC code and/or models for these patterns and practices.

2.2.1. System Coupling through Deployment Strategy

A critical facet of deploying a microservice-based system is to maintain the autonomy and scalability of individual services while promoting loose coupling in the deployment strategy. Services should operate independently, enabling their respective development teams to expedite the process of building and deploying services. Additionally, it's imperative to account for resource consumption per service, as certain services may have limitations on CPU or memory usage. Ensuring availability and behavior monitoring for each independent service is also vital in the deployment process. There are two practices related to deployment strategy:

- **Multiple Services per Execution Environment** In this pattern, services are all deployed in the same execution environment, making it problematic to change, build, and deploy the services independently [17].
- **Single Service per Execution Environment**. This option ensures loose coupling in deployment since each service is independently deployable and scalable, and resource consumption can be constrained for each service and monitoring services separately [17]. Development team dependencies are also reduced.

2.2.2. System Coupling through Infrastructure Stack Grouping

Managing the infrastructure resources can impact significant architectural qualities of a microservice-based system, such as loose coupling between services and independent development in different teams. Grouping different resources into infrastructure stacks should reflect the development teams' responsibilities to ensure independent deployability and scalability. There are two patterns for grouping infrastructure resources into one or multiple stacks.

- **Monolith Stack** All the infrastructure elements are grouped in a single stack in this pattern. Servers, services, containers, and databases are part of a single stack and deployed as a unit [3]. This pattern facilitates adding new elements to a system and stack management. However, there are some risks to using this pattern. Changing a larger application group stack is riskier than changing a smaller one, resulting in more frequent changes. Also, services cannot be deployed and

changed independently, and different development teams may depend on each other.

- **Service Stack** In this pattern, each service has its own stack [3]. Stacks boundaries are aligned with the system and team boundaries. Thus, teams and services are more autonomous and can work independently. Furthermore, services and their infrastructure are loosely coupled since independent deployability and scalability for each service are supported

2.3. Component Models of Infrastructure as Code Systems

In this work, we use and extend a formal model for metrics definition based on our prior work [18]. We extend it here to model the integration of component and deployment nodes. A microservice decomposition and deployment architecture model M is a tuple (N_M, C_M, NT_M, CT_M, c_source, c_target, $nm\_connectors$, n_type, c_type) where:

- N_M is a finite set of component and infrastructure **nodes** in Model M.
- $C_M \subseteq N_M \times N_M$ is an ordered finite set of **connector edges**.
- NT_M is a set of **component types**.
- CT_M is a set of **connector types**.
- $c\_source : C_M \rightarrow N_M$ is a function returning the component that is the **source** of a link between two nodes.
- $c\_target : C_M \rightarrow N_M$ is a function returning the component that is the **target** of a link between two nodes.
- $nm\_connectors : N_M \rightarrow C_M$ is a function returning the set of connectors for a set of nodes: $nm\_connectors(nm) = \{c \in C_M : (\exists n \in nm : (c\_source(c) = n \wedge c\_target(c) \in C_M) \vee (c\_target(c) = n \wedge c\_source(c) \in C_M))\}$.
- $n\_type : N_M \rightarrow NT_M$ is a function that maps each node to its set of **direct and transitive node types**. (for a formal definition of node types see [18]).

- *c_type* : $C_M \rightarrow CT_M$ is a function that maps each connector to its set of **direct and transitive connector types**. (for a formal definition of connector types see [18]).

The connector type *deployed_on* is used to denote a deployment relation of a *Component* (as a connector source) on an *Execution_Environment* (as a connector target). It is also used to denote the transitive deployment relation of *Execution_Environments* on other ones, such as a *Container* is deployed on a *VM* or a *Test_Environment*. The connector type *runs_on* models the relations between execution environments and the devices they run on.

The type *Stack* is used to define deployments of *Devices* using the *defines_deployment_of* relation. Stacks include environments with their deployed components using the *includes_deployment_node* relation.

An IaC-based deployment can be represented as formal component model diagrams. An example of an IaC diagram is shown in Figure 2. The various components and their relations are depicted in such diagrams. In the model, all deployment nodes are of type *Deployment Node*, which has the subtypes *Execution Environment* and *Device*. These have further subtypes, such as *VM* and *Container* for *Execution Environment*, and *Server*, *IoT Device*, *Cloud*, etc. for *Device*.

The connector type *deployed on* is used to denote a deployment relation of a *Component* (as a connector source) on an *Execution Environment* (as a connector target). It is also used to denote the transitive deployment relation of *Execution Environments* on other ones, such as a *Container* is deployed on a *VM*. The connector type *runs on* models the relations between execution environments and the devices they run on.

The type *Stack* is used to define deployments of *Devices* using the *defines_deployment_of* relation. Stacks include environments with their deployed components using the *includes_deployment_node* relation.

The model contains elements from both applications (e.g., *Service, Database*) and infrastructure (e.g., *Container, Infrastructure Stack, Storage Resources*). Furthermore, we have specified all the deployment-related relationships between these elements. In particular, a *Service* and a *Web Server* are *deployed on* a *Container*. A *Database* is *deployed on Storage Resources* and also on a separate *Container*. An *Infrastructure Stack defines deployment of* a *Container* as well as a *Web Server*. All the containers *run on* a *Cloud Server* (e.g., ELK, AWS, etc.).

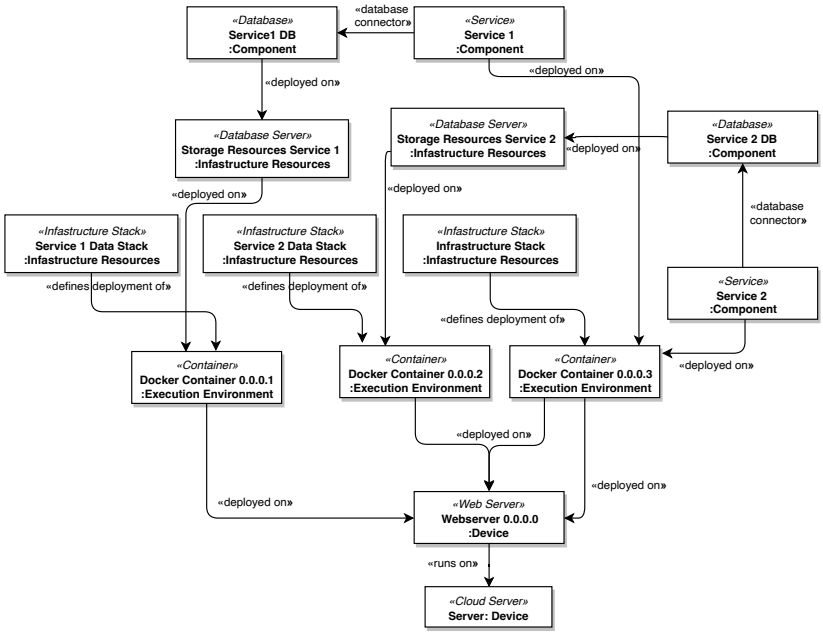

Figure 2: Example IaC-based deployment diagram.

2.4. Metrics

In our previous work [16], we have introduced a set of metrics to measure the support of each IaC coupling practice described in Section 2. The metrics measure each of the coupling-related patterns and practices described above. Metrics in *System Coupling through Deployment Strategy* are a continuous value ranging from 0 to 1, with 0 representing the optimal case where a set of patterns is fully supported and one the worst-case scenario where it is absent. Metrics in *System Coupling through Infrastructure Stack Grouping* have boolean values and return *True* when the corresponding practice is detected.

3. Experiment Planning

Our study adheres to the empirical research guidelines for software engineering proposed by Jedlitschka et al. [19]. In addition, the study design incorporates guidelines for empirical research in software engineering from Kitchenham et al. [20], Wohlin et al. [21], and Juristo and Moreno [22]. For statistical analysis of the collected data, the study uses the robust statisti-

Metric Name	Description
Shared Execution Environment Metric (SEE)	This metric measures the shared execution environments that have service components deployed on them (e.g., a container/VM that two or more services are deployed on) concerning all execution environments with deployed services
Shared Execution Environment Connectors Metric (SEEC)	This metric returns the number of the *shared* direct connectors from deployed service components to execution environments (e.g., containers or VMs) concerning the total number of such service-to-environment connectors. For instance, the connectors of two services deployed on the same container are considered shared.
Monolith Stack Detection Metric (MSD)	The metric returns *True* if only one stack is used in a node-set that deploys more than one service via stacks (e.g., several/all system services are deployed by the only defined stack in the infrastructure) and *False* otherwise.
Service-Stack Detection Metric (SES)	The metric returns *True* is the number of services deployed by stacks equals the number of stacks (e.g., each system service is deployed by its own stack).

Table 1: Metrics for System Coupling

cal method guidelines for empirical software engineering by Kitchenham et al. [23].

3.1. Goals

This experiment aims to conduct a code review of two medium-sized IaC deployment architectures within a limited 1.5-hour session, focusing on the utilization of recommended coupling design patterns and practices. The primary emphasis is on examining specific coupling design patterns and practices that have been recommended. Specifically, we will delve into the System Coupling through Deployment Strategy and System Coupling through Infrastructure Stack Grouping practices as previously elucidated. Participants were provided with a system source code repository for both systems, complete with concise documentation in a README file. To evaluate the extent to which models and metrics can assist participants in comprehending a given system, each participant received guidance through a model of the

IaC system and associated metrics for one of the two tasks. In tasks where this assistance was available, it was suggested that participants consult the models and metrics first and then delve into the code for more detailed insights. The metrics and models employed in this study are based on the methods introduced in the paper [16].

3.2. System Description

The selection of the systems is based on their distinctive approaches and technologies used for deploying and managing cloud-based services, focusing on monitoring, deployment, and their strong representation of the coupling-related practices investigated in our approach. We selected them as they represent major approaches regarding the two studied ADDs, as identified in our study of gray literature sources and an in-depth analysis of open-source systems from our prior work [16]. Table 2 shows the technologies used in the selected systems.

Each IaC technology used in these systems presents different levels of complexity and understandability:

- **Docker Compose:** Docker Compose is relatively straightforward and designed for local development and testing. It provides a simple YAML syntax for defining multi-container applications, making it easier to understand and use, even for developers with minimal infrastructure experience.

- **Kubernetes Manifests:** Kubernetes configurations are significantly more complex than Docker Compose files. Kubernetes requires a deep understanding of clusters, services, networking, and resource allocation. As observed in our experiment, the **higher level of abstraction in Kubernetes manifests often required participants to rely on additional resources such as models and metrics to correctly interpret deployment details.

- **Terraform and Cloud Deployment Scripts:** While our selected systems primarily use Kubernetes and Docker-based IaC, cloud-based infrastructure provisioning tools like Terraform introduce a different layer of complexity. Terraform enables declarative infrastructure management across multiple cloud providers but does not explicitly define service relationships the way Kubernetes does. This means that while

Terraform excels at provisioning infrastructure, it may require additional context for understanding service interdependencies, which is critical for evaluating coupling-related practices.

E-shop on Containers (ESC). [1]. The *eShopOnContainers* is a sample reference application developed by Microsoft, based on a microservices architecture and Docker containers, designed to run on Azure and Azure cloud services. It features multiple autonomous microservices and supports various communication styles (e.g., synchronous and asynchronous via a message broker). The application also includes files necessary for deployment on a Kubernetes cluster and provides the required IaC scripts for integrating with ELK (Elasticsearch, Logstash, Kibana) for logging purposes.

Sock Shop Application (SSA). [2]. The *Sock Shop* is a reference application for microservices created by Weaveworks to demonstrate microservice architectures and related technologies. It illustrates cloud-native technologies, using *Kubernetes* for container orchestration, with services deployed on *Docker* containers. *Terraform* infrastructure scripts are included to deploy the system on *Amazon Web Services (AWS)*. We believe this application is a good representative of current industry practices in microservice-based architectures and IaC deployments.

Table 2: Infrastructure as Code Technologies Used in the Selected Systems

System Name	**IaC Technologies Used**	**Purpose**
E-shop on Containers (ESC)	Kubernetes manifests, Azure deployment scripts, ELK Stack configurations, Windows-based deployment	Deploys microservices on **Kubernetes**, supports **Azure-based provisioning**, and includes **logging infrastructure** (ELK).
Sock Shop Application (SSA)	Docker, Kubernetes manifests	Supports **local development with Docker Compose** and **cloud deployment using Kubernetes**.

We deliberately chose relatively small but realistic systems to avoid potential bias caused by fatigue effects in our controlled experiment. Smaller

[1] `https://gitlab.swa.univie.ac.at/ntentose84swa/e-shop-on-containers`
[2] `https://gitlab.swa.univie.ac.at/ntentose84swa/sock-shop-application`

systems also help minimize the learning curve for participants, especially if they are unfamiliar with the specific IaC tools used in the experiment. Additionally, participants will likely remain more motivated when tasks can be completed within a reasonable timeframe. It's also worth noting that, in larger systems, IaC components are typically modularized. This means that the portion of the IaC code necessary to evaluate a particular security ADD is not larger than the example IaC scripts used in our study. However, identifying the relevant IaC scripts in larger systems is usually more difficult. We specifically chose smaller systems to assess the understandability of a given script rather than testing participants' ability to locate the correct IaC script.

3.3. Context and Design

We randomly divided the participants into A1, A2, B1, and B2 groups. We used a Within-Subjects Design [24] in which the same person tests all the conditions. Each participant got two tasks, one with help and one without. The two sub-tasks are called:

- E-shop on Containers (ESC)
- Sock Shop Application (SSA)

The groups are composed as follows:

- A1) ESC: system source code repository (formal models/metrics) + SSA: only system source code repository
- A2) SSA: only system source code repository + ESC system source code repository (formal models/metrics) – identical to A1, only order reversed
- B1) SSA: system source code repository (formal models/metrics) + ESC: only system source code repository
- B2) ESC: only system source code repository + SSA: system source code repository (formal models/metrics) – identical to B1, only order reversed

We added the reserved-order groups A2 and B2 to avoid possible biases from the tasks with or without help. For example, participants might spend

more time on the first task, or there could be some learning effect in the general code-reviewing activity. Henceforth, we call the tasks in the four groups, where the participants get only a system source code repository, the Control Tasks (CT), and the tasks in the four groups, where the participants get a system source code repository plus help in the form of formal metrics and models, the Experiment Tasks (ET).

3.4. Participants

Our participants were Master's students in the Advanced Software Engineering (ASE) course. These students were allocated 5% of their course points as a reward for participating in this experiment, which was integrated into their lecture as a hands-on task. Beyond this academic incentive, no additional rewards or incentives were offered. Students could participate in the lecture task but were not obliged to participate in the experiment (opt-out option).

In the lectures preceding this hands-on task, we covered topics related to software design, domain-driven design, software architecture, software patterns and practices, microservice architectures, and DevOps. Participants were provided with an introductory text that familiarized them with background information on IaC, relevant IaC coupling practices, the modeling techniques employed in the experiment, and the metrics used.

Our study is not centered on techniques applicable only to a select group of highly trained experts. Instead, we aimed to assess the utilization of these techniques by software engineers who may not be experts in all facets of the study or even novice software engineers. Following the findings of Kitchenham et al. [20], our choice of students as participants is appropriate, as they represent the upcoming generation of software professionals and thus align with the population of interest. This is further substantiated by the fact that some students who participated in our experiment possessed several years of programming and industry experience. Additionally, other studies, such as those conducted by Svahnberg et al. [25] and Salman et al. [26], argue that students can effectively serve as valid representatives for professionals in specific empirical software engineering experiments.

3.5. Material and Tasks

The experiment is based on a selection of coupling-related practices in IaC. The selection includes the practices for *System Coupling through Deployment Strategy* and *System Coupling through Infrastructure Stack Group-*

ing summarized in Section 2. The selected software practices are related to the subjects taught in the course ASE, such as related software modeling techniques, software design patterns, and software architecture. This study consists of two major experiment material artifacts[3]:

- **(1) Information Sheet.** A document explaining the IaC coupling practices with an example model and the related metrics was provided to participants two weeks before the experiment was conducted.
- **(2) Survey Form.** Four experiment survey forms per group were handed out to participants during the experiment.

All survey forms for experiments follow a consistent structure, comprising three main components: (1) a questionnaire to gather participant information, (2) two experimental tasks specifically designed for one of the four groups (A1, A2, B1, B2, as elaborated in Section 3.3), and (3) an overall experiment questionnaire.

Each task is subdivided into sub-tasks to evaluate participants' comprehension of coupling-related practices. Participants were first instructed to review the code and descriptions in the designated repositories for each system. Afterward, they were directed to address the following six sub-tasks:

- Two tasks on listing model elements (components and connectors) were used to determine the understanding of system structure and relations. An example question in Task ESC.3 is: *"List the Docker-based execution environments on which the Ordering, Basket, Identity, and Catalog services are deployed and the database servers they use."*
- Two tasks with three filling-out-blanks sentences were used to determine the understanding of system structure. An example sentence in Task ECS.2: *"The application uses ___ number of infrastructure stacks that define the deployment of the Ordering, Basket, Identity, and Catalog services."*
- Two tasks with three True/False answers of practices were used to determine the understanding of used/supported/realized techniques in the provided systems. An example in Task SSA.1:

[3]See the experiment documents provided online in a long-term archive to enable replicability of our study at `https://doi.org/10.5281/zenodo.14892817`

- ☐ *The User and Order services are grouped in different infrastructure stacks.*
- ☐ *The Catalog and User services are both grouped in the same infrastructure stack.*
- ☐ *The User and Order services are deployed on separate Docker-based execution environments.*

- In addition to the content tasks, a task-based questionnaire was used to obtain an objective perspective of the participants' self-assessment of how to correct their answers with a certain level of confidence.

Table 3 presented above outlines the main structure of the questionnaire used in the hands-on tasks.

3.6. Variables and Hypotheses

In this study, we define correctness and duration as dependent variables to assess participants' ability to understand coupling-related practices in IaC-based deployments. Correctness measures the effectiveness of understanding by evaluating how accurately participants complete the assigned tasks, while duration represents the efficiency of understanding by measuring the time taken to complete the tasks.

We consider duration as a dependent variable where task completion times can indicate increased cognitive effort. However, longer duration does not necessarily equate to improved understandability. A participant may take longer to complete a task due to difficulties in comprehension, lack of familiarity with the materials, or additional cognitive effort required to analyze complex structures. Conversely, shorter durations could indicate either efficient understanding or superficial processing. To better capture this dynamic relation, we analyze the relationship between correctness and duration across different conditions.

We devised the following Null Hypotheses:

- **$H_0$1** There is no significant difference in *correctness* for *Experimental* compared to *Control*.
- **$H_0$2** There is no significant difference in *duration* for *Experimental* compared to *Control*.

Table 3: Structure of the Questionnaire

Section	Questions/Elements
Demographics	Age (Optional), Gender (Optional), Highest Education Level, Programming Experience, Modelling Experience, Industry Experience, Prior Knowledge in Infrastructure as Code, Formal Modelling of Systems
Hands-On Task (ESC)	ESC.1 - System Deployment Verification, ESC.2 - Infrastructure Stack Identification, ESC.3 - Execution Environment Mapping, ESC.4 - Component Deployment Verification, ESC.5 - Infrastructure Deployment Description, ESC.6 - Execution Environment Count
ESC Survey	Confidence in answers, Ease of understanding textual system descriptions, Ease of understanding system diagrams, Identifying components and relations, Understanding coupling-related practices, Perceived usefulness of metrics and models for large-scale systems, Understanding the concept of Infrastructure Stacks
Hands-On Task (SSA)	SSA.1 - Infrastructure Stack Grouping, SSA.2 - Execution Environment Mapping, SSA.3 - Infrastructure Stack and Database Deployment, SSA.4 - Service Grouping, SSA.5 - Execution Environments and Infrastructure Stacks, SSA.6 - System Deployment Verification
SSA Survey	Confidence in answers, Ease of understanding textual system descriptions, Ease of understanding system diagrams, Identifying components and relations, Understanding coupling-related practices, Perceived usefulness of metrics and models for large-scale systems, Understanding the concept of Infrastructure Stacks
Concluding Survey	Familiarity with textual and graphical descriptions before the study

- **$H_0$3** There is no significant increase in *correctness* as *duration* increases for *Experimental* compared to *Control*.

and the corresponding Alternative Hypotheses:

- **H_a1** There is significant difference in *correctness* for *Experimental* compared to *Control*.
- **H_a2** There is significant difference in *duration* for *Experimental* compared to *Control*.
- **H_a3** There is significant increase in *correctness* as *duration* increases for *Experimental* compared to *Control*.

While correctness is expected to improve with supplementary models and metrics, the effect on duration remains uncertain due to potential confounding factors such as task complexity, prior experience, and cognitive load. Thus, we further analyze the interaction between correctness and duration in Section 6.3.

Figure 3 illustrates a visual research model summarizing how self-confidence and prior knowledge could act as mediator variables influencing how models affect correctness, while experience with programming and modeling could serve as moderator variables, impacting the extent to which participants benefit from models.

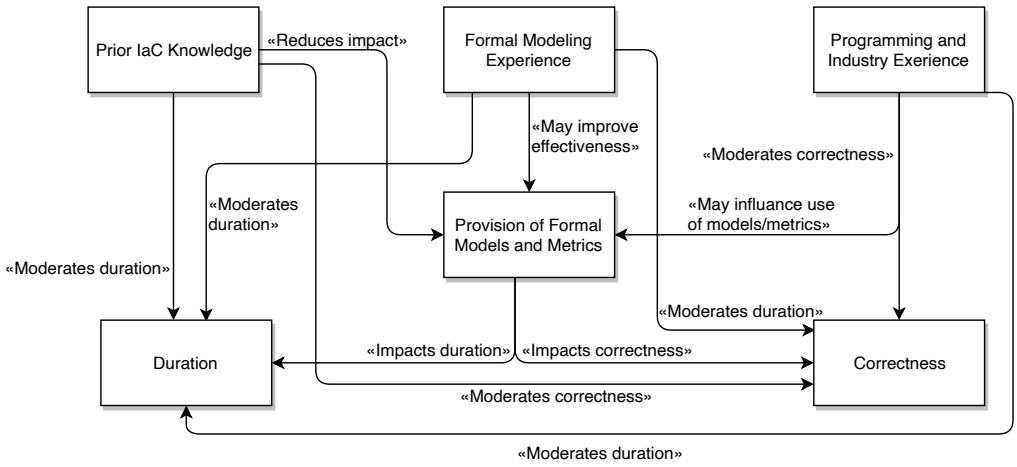

Figure 3: Research Model Diagram.

4. Experiment Execution

This experiment was executed in two steps: a preparation and a procedure phase.

4.1. Preparation

Two weeks before the experiment, we distributed the preparation material, which is the experiment information sheet (refer to Section 2), through an e-learning platform[4]. This document served as a comprehensive resource, offering general information about the upcoming experiment and an introduction to IaC technologies. It thoroughly explained all coupling-related patterns and practices, a set of metrics, an illustrative model, and detailed descriptions. Participants were granted permission to have a printed copy of this document during the experiment. The provision of the experiment information document was essential to ensure that all participants were uniformly educated about coupling practices in IaC-based deployments, as detailed in Section 2.

4.2. Procedure

The experiment was conducted using traditional pen-and-paper methods and personal computers (PCs) with restricted access for navigating the source code repositories of the two IaC systems. Participants were permitted to bring only the preparation material, as outlined in the previous Section 3, to assist them in completing the experiment survey form. At the beginning of the experiment, each participant was given a randomly selected experiment survey form (see Section 3.5). To ensure fairness, an approximately equal number of each form type (A1, A2, B1, B2, as detailed in Section 3.3) was distributed during this random allocation process. Participants were given clear instructions to complete the survey in the specified order, starting from the first page and progressing to the last.

Additionally, a clock with second-level granularity was displayed to offer timestamp information to participants. They were tasked with recording both the start and stop timestamps while working on the experiment tasks. These timestamps were later converted into durations in seconds and then summed to calculate the total duration for all tasks. To safeguard the confidentiality of participant data, an individual who was not directly involved in

[4] `https://moodle.univie.ac.at/`

the experiment was responsible for dissociating personal information (such as names and student numbers) from the experiment sheets by assigning a unique identification number.

5. Analysis

The statistical analysis was conducted with the R software tool[5]. The analysis consists of multiple steps: loading the pre-processed dataset (Section 5.1), calculating descriptive statistics for the dependent variables (Section 5.4), performing group-by-group comparisons utilizing appropriate statistical hypothesis tests (Section 5.5) and generating tables and plots that were included in the article. To reproduce these results, it is necessary to install the specific R library package dependencies[6].

5.1. Data-Set Preparation

The collected raw data[7] from the experiment execution phase (refer to Section 4) was prepared as follows: (1) the results from the experiment sheets were reported in a Comma-Separated Values (CSV) file; (2) the CSV file was imported in RScript for type casting and filtering of the rows with missing time information; and (3) the correctness for the task as well as each sub-task was calculated. The data set is published in the long-term open data archive Zenodo[8] with all documents and R scripts.

5.2. Participant Demographics

Participants exhibited diverse demographic profiles regarding ages (Figure 4), educational backgrounds (Figure 9), industry experiences (Figure 6), and programming experience (Figure 5). The age of participants mainly varied from 20 to 40 years, indicating a mix of early-career to mid-career individuals, with a few older ones. In terms of education (Figure 9), the experiment included individuals with Bachelor of Science (BSc) and Master of Science (MSc) degrees. Experience with formal modeling (Figure 8) was balanced between participants with and without this experience, while

[5]See `https://www.r-project.org` for version 4.2.2.
[6]See install.r in the replication package.
[7]See experiment_results.csv in the replication package.
[8]`https://doi.org/10.5281/zenodo.14892817`

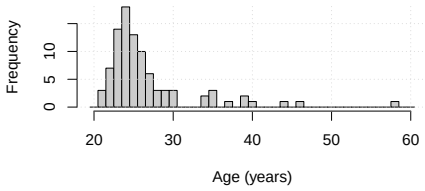

Figure 4: Participants' Age.

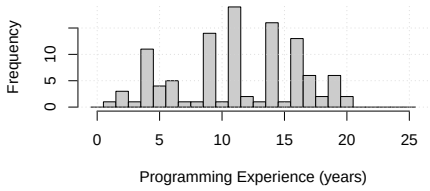

Figure 5: Participants' Programming Experience.

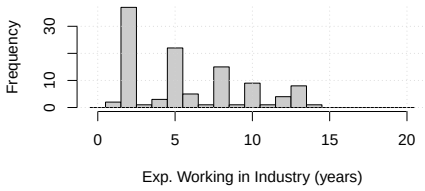

Figure 6: Participants' Software Industry Experience.

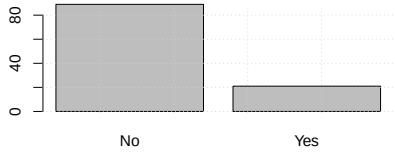

Figure 7: Participants' Infrastructure as Code Experience.

experience with IaC (Figure 7) tended towards inexperience. The industry experience ranged from 0 to 15 years, with the distribution indicating a higher proportion of early-career professionals compared to those with more experience. Finally, programming experience also showed a wide range, from 0 to more than 20 years, highlighting a wide range of technical proficiency.

The participants in this study exhibited diverse backgrounds in terms of age, education, programming experience, industry experience, and prior familiarity with IaC. These factors may have played a moderating role in the relationship between the independent variable (availability of formal models and metrics) and the dependent variables (*correctness* and *duration*).

5.3. Normality Assessment

The normal Q-Q plots of *correctness* for both ESC (Figure 10) and SSA

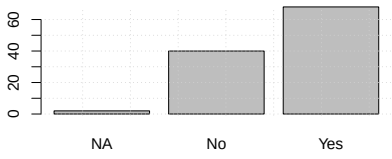

Figure 8: Participants' Formal Modelling Experience.

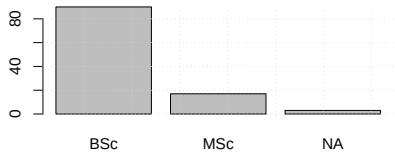

Figure 9: Participants' Education.

(Figure 11) appear to indicate a normal distribution for *Control* but not for *Experimental*.

We use Shapiro-Wilk [27] to test normality. According to Razali and Yap [28], it is more powerful than Anderson-Darling [29], Lilliefors [30], Kolmogorov-Smirnov [31], and other alternatives. For $\alpha = 0.05$, the test for *Control* indicated that the distributions are not significantly different from the normal distribution, with a $p > \alpha$ values for both ESC ($p = 0.5586$) and SAA ($p = 0.0880$). However, for *Experimental*, the test indicated that the distributions significantly deviate from the normal distribution, whereas $p \leq \alpha$ for both ESC ($p = 0.0041$) and SAA ($p = 0.0015$).

Visual inspection of the normal Q-Q plots for both groups and both systems for *duration*, visible in Figures 12 and 13, was insufficient to determine whether each group's data were normally distributed. Here, the Shapiro-Wilk normality test returned $p \leq \alpha$ for all groups and systems, meaning no distribution was normal.

5.4. Descriptive Statistics

5.4.1. Correctness

Table 4 and Table 5 show the number of corresponding observations, central tendency measures, and dispersion measures per group for the dependent variable *correctness*[9]

[9] *correctness* is defined as a value in $[0, 1] \cap \mathbb{R}$.

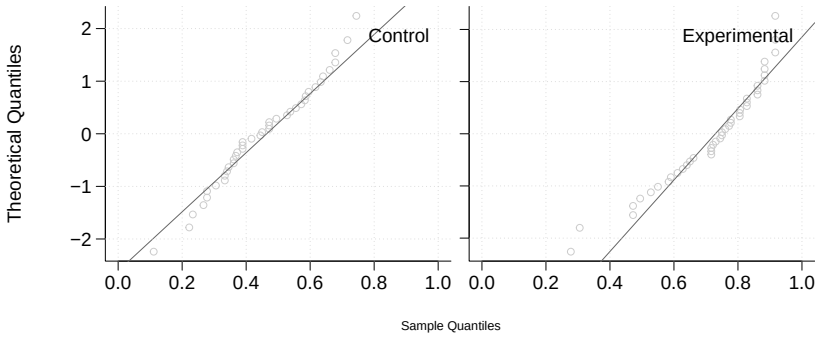

Figure 10: Normal Q-Q Plot of *Correctness* (ESC)

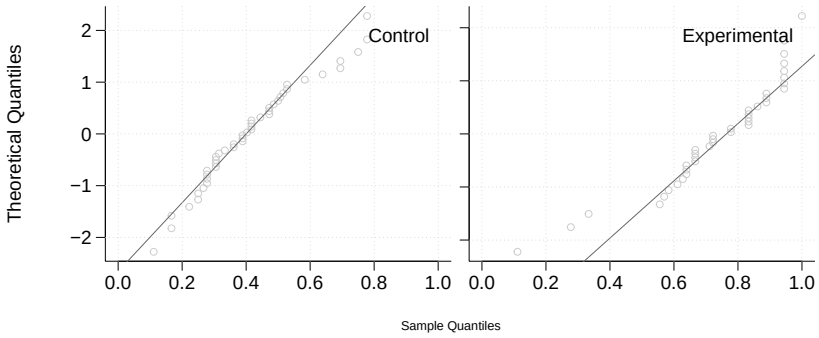

Figure 11: Normal Q-Q Plot of *Correctness* (SSA).

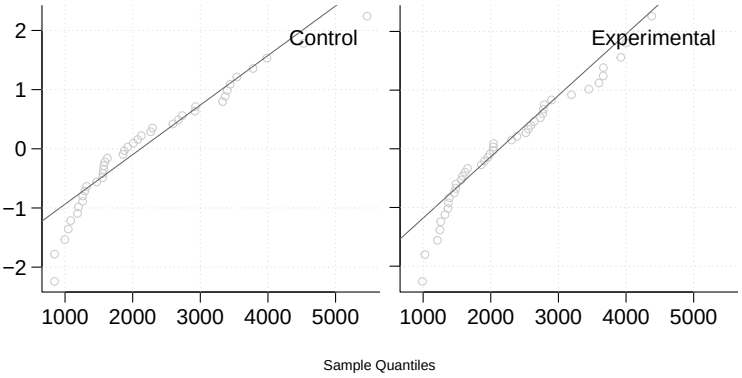

Figure 12: Normal Q-Q Plot of *Duration* (ESC).

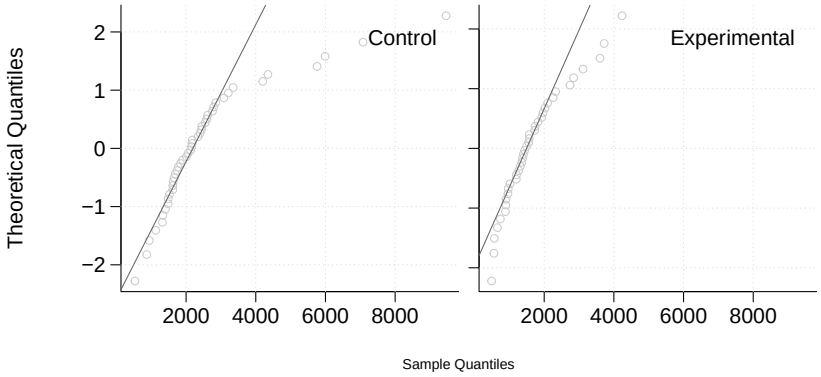

Figure 13: Normal Q-Q Plot of *Duration* (SSA).

Table 4: Descriptive Statistics per Group of Dependent Variable *Correctness* (ESC)

	Control	*Experimental*
Number of observations	40	42
Mean	0.4556	0.7183
Standard deviation	0.1538	0.1585
Median	0.4472	0.7500
Median absolute deviation	0.1689	0.1637
Minimum	0.1111	0.2778
Maximum	0.7444	0.9167
Skew	0.0204	-0.9507
Kurtosis	-0.9077	0.3551
Shapiro-Wilk Test p-value	0.5586	0.0041

Table 5: Descriptive Statistics per Group of Dependent Variable *Correctness* (SSA)

	Control	*Experimental*
Number of observations	44	38
Mean	0.4140	0.7366
Standard deviation	0.1638	0.1968
Median	0.3958	0.7500
Median absolute deviation	0.1596	0.1730
Minimum	0.1111	0.1111
Maximum	0.7778	1.0000
Skew	0.5522	-1.1409
Kurtosis	-0.3061	1.3388
Shapiro-Wilk Test p-value	0.0880	0.0015

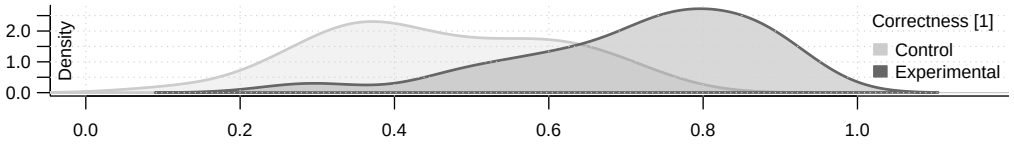

Figure 14: Kernel Density Plot of *Correctness* (ESC).

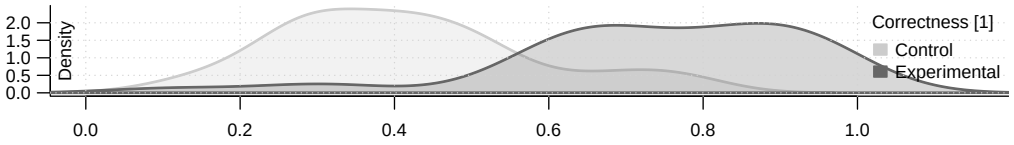

Figure 15: Kernel Density Plot of *Correctness* (SSA).

These statistics are illustrated as a kernel density plot in Figure 14 and Figure 15.

By visually inspecting the correctness results (Figures 14 and 15) and examining the skew values in Tables 4 and 5, it can be observed that the distribution of *Control* appears to be symmetrical to moderately positively skewed, while the distribution of *Experimental* exhibits a negative skew. The excess kurtosis values are negative for *Control* and positive for *Experimental*, indicating longer-tailed distribution for *Control*.

5.4.2. Duration

Table 6 and 7 shows the number of observations, central tendency measures, and dispersion measures per group for the dependent variable *duration*[10]

Regarding duration results, the skews shown in Tables 6 and 7 indicate that *Control*'s and *Experimental*'s distributions are positively skewed, with *Experimental*'s distributions being closer to symmetry. The excess kurtosis values are similarly varied but lower for *Experimental* than for *Control*.

[10] *Duration* is denoted in seconds.

Table 6: Descriptive Statistics per Group of Dependent Variable *Duration* (ESC)

	Control	*Experimental*
Number of observations	40	42
Mean	2209.90	2255.05
Standard deviation	1100.13	908.09
Median	1901.50	2038.50
Median absolute deviation	1034.11	974.81
Minimum	844.00	990.00
Maximum	5466.00	4380.00
Skew	0.9218	0.5879
Kurtosis	0.2019	-0.7478
Shapiro-Wilk Test p-value	0.0043	0.0162

Table 7: Descriptive Statistics per Group of Dependent Variable *Duration* (SSA)

	Control	*Experimental*
Number of observations	44	38
Mean	2556.55	1673.18
Standard deviation	1702.32	906.84
Median	2143.50	1466.50
Median absolute deviation	843.60	745.01
Minimum	532.00	490.00
Maximum	9463.00	4235.00
Skew	2.1720	1.0595
Kurtosis	5.1390	0.5469
Shapiro-Wilk Test p-value	0.0000	0.0036

5.5. Hypothesis Testing

5.5.1. Correctness *and* Duration

To select the appropriate statistical test, we compare the variances of both groups for both dependent variables to assess their equality. We opted for Cliff's δ[32] as a robust and nonparametric test recommended by Kitchenham [33] for scenarios where data distribution, differing distributions between populations, or unequal variances are present. It estimates the probability that a randomly selected observation from one group is larger than a randomly selected observation from another group, taking into account the reverse probability [34]. Cliff's δ was originally designed for measuring ordinal data, but it is equally applicable to the quantitative and continuous data used in this study [35, 36].

One important aspect of multiple hypothesis tests is the necessity to adjust the significance level (α) to mitigate the risk of Type I errors[11]. Of the diverse existing methods, such as the false discovery rate [37] or the Bonferroni-Dunn [38, 39] correction, we opted for Bonferroni-Dunn correction as the most stringent form of correction. It is calculated using Equation 1:

$$\alpha' = \frac{\alpha}{n} \tag{1}$$

with n being the number of times a test is applied. This results in $\alpha' = \frac{0.05}{2} = 0.025$ where $\alpha = 0.05$ and $n = 2$ in our case.

The results of the one-tailed Cliff's δ test are shown in Tables 8 and 9 for

[11] This is not required for normality or comparing variances.

Table 8: Hypothesis Tests per Group Combination of the Dependent Variable *Correctness* (ESC)

	Control vs. *Experimental*
Cliff's δ Test	
Cliff's δ	0.7607
s_δ	0.0749
v_δ	0.0056
z_δ	10.1584
CI_{low}	0.5725
CI_{high}	0.8728
$P(X > Y)$	0.1155
$P(X = Y)$	0.0083
$P(X < Y)$	0.8762
p	0.0000

Table 9: Hypothesis Tests per Group Combination of the Dependent Variable *Correctness* (SSA)

	Control vs. *Experimental*
Cliff's δ Test	
Cliff's δ	0.7715
s_δ	0.0812
v_δ	0.0066
z_δ	9.4997
CI_{low}	0.5613
CI_{high}	0.8882
$P(X > Y)$	0.1112
$P(X = Y)$	0.0060
$P(X < Y)$	0.8828
p	0.0000

correctness and Tables 10 and 11 for *duration*.

For *correctness*, Cliff's δ indicates by $p \leq \alpha'$ that *Experimental* scored significantly higher than *Control*. For *duration*, Cliff's δ yielded $p > \alpha'$ for ESC, so we cannot conclude that *Control* took significantly longer than *Experimental* to complete the experiment. The negative Cliff's δ for SSA's *duration* indicates that, on average, the *Experimental* (which received additional IaC diagrams and metrics) completed the tasks in less time than the control group.

Alternatively, employing the Multivariate Analysis of Variance (MANOVA) statistical test would have been conventional for comparing multiple groups using several dependent variables under specific conditions [40], including for all distributions to be normal. The test aims to decide whether independent variables impact the dependent variables, individually or in combination. Per Section 5.4, the distributions of *Control* for *correctness* do not significantly differ from the normal distribution. However, the distributions of *Experimental* for *correctness* differ from the normal distribution, as well as those of both groups for *duration*, which is why we have decided against this test.

Based on Cliff's δ tests for both ESC and SSA, concerning *correctness*, we can reject the null hypothesis $\mathbf{H_0 1}$ and thus accept the alternative hypothesis $\mathbf{H_a 1}$. Conversely, concerning *duration*, we are not able to reject the null hypothesis $\mathbf{H_0 2}$ and thus must reject the alternative hypothesis $\mathbf{H_a 2}$, since the result was inconclusive for ESC.

Table 10: Hypothesis Tests per Group Combination of the Dependent Variable *Duration* (ESC)

	Control vs. *Experimental*
Cliff's δ Test	
Cliff's δ	0.0798
s_δ	0.1291
v_δ	0.0167
z_δ	0.6178
CI_{low}	-0.1740
CI_{high}	0.3236
$P(X > Y)$	0.4595
$P(X = Y)$	0.0012
$P(X < Y)$	0.5393
p	0.5385

Table 11: Hypothesis Tests per Group Combination of the Dependent Variable *Duration* (SSA)

	Control vs. *Experimental*
Cliff's δ Test	
Cliff's δ	-0.4115
s_δ	0.1163
v_δ	0.0135
z_δ	-3.5381
CI_{low}	-0.6123
CI_{high}	-0.1608
$P(X > Y)$	0.7057
$P(X = Y)$	0.0000
$P(X < Y)$	0.2943
p	0.0007

5.5.2. Correlation Between Correctness *and* Duration

We then visually examined the scatter plot in Figures 16 and 17, which aim to explore potential correlations between the two dependent variables *correctness* and *duration*. While ESC's *Control* might show a moderate linear correlation, no evident linear correlation was observed in the other diagrams.

Following the visual inspection, we perform a correlation test with Spearman's ρ test. As shown in Tables 12 and 13, only ESC's *Control* reveals a significant correlation, indicated by $p \leq \alpha'$, demonstrating a moderate positive association between *correctness* and *duration*. We are therefore not able to reject the null hypothesis $\mathbf{H_0 3}$ for either test or group and must reject the alternative hypothesis $\mathbf{H_a 3}$.

5.6. Observations

As explained in Section 3.5, participants were instructed to fill out a survey after each task to evaluate their confidence level in the accuracy of their responses and their understanding of system descriptions, semi-formal models, and metrics. The self-assessment score is calculated using a formula based on participants' answers' correctness (*correctness*) and confidence level, with Equation 2:

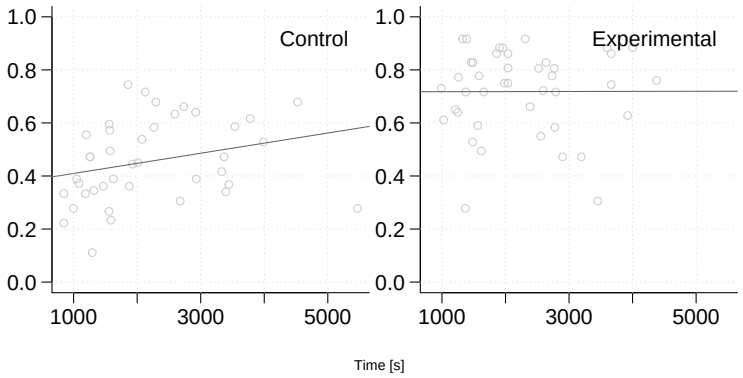

Figure 16: Scatter Plot per Group of the Dependent Variables *Correctness* to *Duration* (ESC).

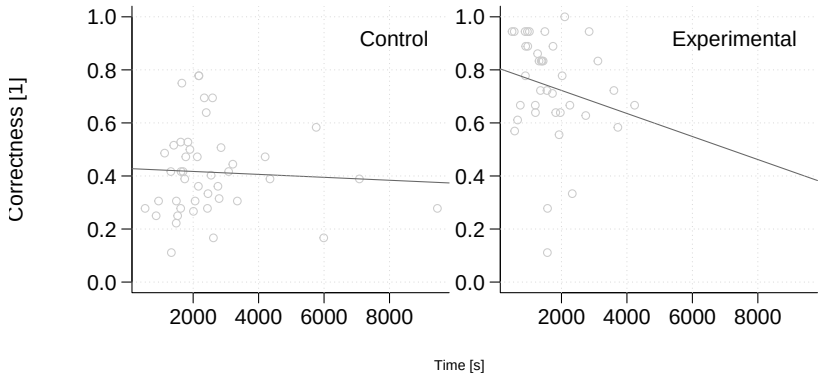

Figure 17: Scatter Plot per Group of the Dependent Variables *Correctness* to *Duration* (SSA).

Table 12: Correlation per Group of the Dependent Variables *Correctness* with *Duration* per Group (ESC)

	Control	*Experimental*
Spearman's ρ	0.3930	0.0253
p	0.0121	0.8736
S	6471.0349	12028.6328

Table 13: Correlation per Group of the Dependent Variables *Correctness* with *Duration* per Group (SSA)

	Control	*Experimental*
Spearman's ρ	0.1167	-0.2829
p	0.4506	0.0853
S	12533.8979	11724.2099

$$self_{assessment} = self_{correctness} - \frac{5 - self_{confidence}}{5} \quad (2)$$

where $self_{correctness}$ represents the participants' average *correctness* as defined in 5.4. 0 indicates entirely incorrect answers, and 1 means completely correct answers.

$self_{confidence} \in [0, 5] \cap \mathbb{R}$ represents the participant's average confidence based on a survey that uses a five-point Likert scale. Each point on the scale is assigned equidistant values within the range of 1 to 5. A value of 1 signifies high confidence in the correctness of the answers, while a value of 5 indicates low confidence.

$self_{assessment} \in [-1, 1] \cap \mathbb{R}$ represents the average self-assessment score of participants. A value less than 0 ($self_{assessment} < 0$) indicates that participants tend to overestimate the correctness of their answers. A value of 0 ($self_{assessment} = 0$) indicates that participants accurately estimate the correctness of their answers. A value greater than 0 ($self_{assessment} > 0$) indicates that participants tend to underestimate the correctness of their answers.

Figure 18 illustrates the kernel density plot of participants' overall self-assessment score per group. The plot shows that, on average, participants underestimated their *correctness* across all tests and groups and that this trend was slightly more marked for *Experimental*.

Visually complementing the statistical data, Figures 20 and 21 represent the distribution of understanding scores for system descriptions by participants in the *Control* and *Experimental*. Figures 22 and 23 illustrate the distribution of understanding scores for semi-formal models and metrics. These histograms showcase the frequency of scores across a range from 1 to 5, with clear distinctions between the two groups.

The *Control*'s distribution is more concentrated towards higher scores, aligning with the higher mean and median values noted. This visualization supports the statistical finding of better self-assessed system comprehension in the *Control*.

6. Discussion

6.1. Correctness

The alternative hypothesis H_a1 was accepted, demonstrating that using models had the anticipated effect on *correctness*. The visual inspection of Kernel Density plots reveals two distinct distributions, with the Experiment

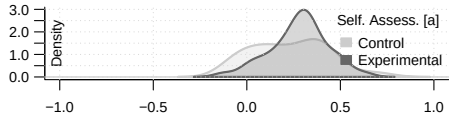

Figure 18: Kernel Density Plot per Group of
Participants' Self Assessment (ESC).

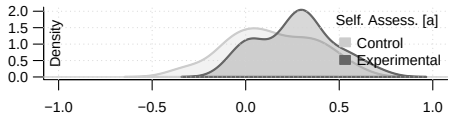

Figure 19: Kernel Density Plot per Group of
Participants' Self Assessment (SSA).

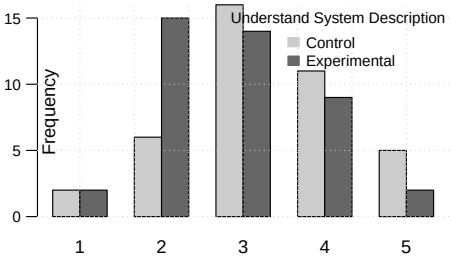

Figure 20: Histogram per Group of Participants' Understanding
of system descriptions (ESC).

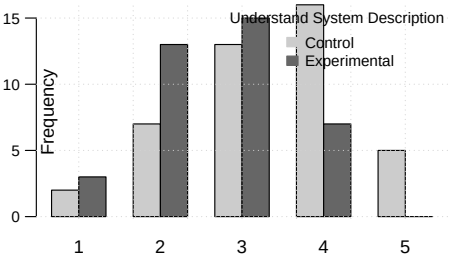

Figure 21: Histogram per Group of Participants' Understanding
of System descriptions (SSA).

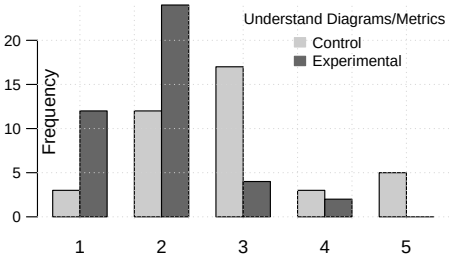

Figure 22: Histogram per Group of Participants' Understanding
of semi-formal models and metrics (ESC).

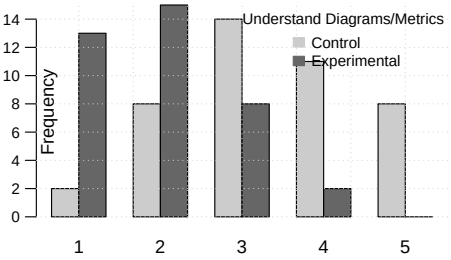

Figure 23: Histogram per Group of Participants' Understanding
of semi-formal models and metrics (SSA).

group scoring higher than Control. Cliff's delta test confirms that this visual inspection is statistically significant and that participants did provide more accurate answers when aided by a model. Per our initial expectation, including semi-formal models and metrics significantly improves the comprehension of architecturally significant coupling best practices in IaC.

However, while models significantly improved correctness, they did not completely eliminate the need for participants to reference the IaC code. Although participants with access to semi-formal models and metrics achieved better comprehension of coupling-related practices, they still consulted the source code to verify specific configuration details, dependencies, and low-level implementation aspects that were not explicitly represented in the models. This suggests that the models effectively provided a higher-level abstraction that facilitated understanding but did not serve as a complete substitute for code-based inspection.

A limitation of our study is that it does not distinguish whether the correctness improvements were driven primarily by the models, the metrics, or their combined effect. While the models provided structured visual representations of system architecture, the metrics quantitatively highlighted coupling-related practices, and participants may have benefited differently depending on their familiarity with each representation type. Some participants might have relied more on models due to their clarity in illustrating dependencies, while others may have found metrics more useful for pinpointing specific coupling issues.

6.2. Duration

In contrast to our expectations, we had to reject H_a2 in favor of the null hypothesis H_02. The visual inspection of Scatter Plots does not reveal any trend about *duration*. Cliff's delta test showed a negative correlation for SSA, as we expected to observe, but it did not for ESC. While we expected to observe such a reduction in *duration* for the participants using a model, we could not establish it for the general case.

One explanation is that participants may have spent additional time analyzing the supplementary models and metrics before engaging with the source code, leading to a compensatory effect where any efficiency gains were offset by the need to interpret multiple representations of the system. Another factor is task complexity—some participants may have taken longer simply due to the inherent difficulty of certain tasks rather than because they were struggling with comprehension. Additionally, individual differences in prior

experience with modeling, programming, and IaC concepts may have influenced how quickly participants processed the provided materials.

Alternatively, our reliance on self-reported duration by the participants may be faulty and inadequate for measuring the phenomenon. Many participants rounded up their timestamp to the minute, making any effect on duration undetectable for the shorter sub-tasks. If that were the case, it would be an argument favoring ESC being simpler than SSA.

Further experiments on this topic should focus on decisively establishing the effect of models on *duration* or the lack thereof. Having more precise time measurements down to the second for short tasks would be ideal. However, it may be impractical in an experimental setting like ours, where many participants self-report timestamps and cannot be monitored individually. Another approach would be to submit longer tasks to the participants to make the time difference more apparent.

6.3. Correctness and Duration

A key question in this study was whether increased duration correlates with improved correctness, which would suggest a deeper level of comprehension. The results indicate that while such a correlation was observed in the control group for the ESC system, it was not consistently present across other conditions. This suggests that spending more time on a task does not universally lead to better correctness.

In the ESC *Control*, a moderate positive correlation was found between *correctness* and *duration*, indicating that participants who spent more time on the task tended to provide more accurate answers. However, no such correlation was found in the experimental groups, where participants had access to formal models and metrics. This discrepancy suggests that the additional materials may have allowed participants to reach correct answers without necessarily increasing the time spent on tasks.

Additionally, the absence of a strong correlation in other cases implies that task complexity, cognitive effort, and prior knowledge may have played a greater role in correctness than duration alone. Some participants may have completed tasks quickly due to familiarity with infrastructure-as-code and modeling concepts, while others may have struggled with the materials regardless of the time spent.

One point worth noting is that the improvement in *correctness* observed with the aid of models did not come at the cost of slower performance, suggesting that the benefits of semi-formal models and metrics are not coun-

terbalanced by a deterioration in the time required to complete tasks, which is a point in their favor.

6.4. Observations

A slight but pervasive trend in the participants' self-assessment is that participants in the experimental groups are consistently less confident than control group participants. Control group participants assessed their confidence and understanding at a higher level than those in the experimental groups. While it was not part of our hypotheses, this pattern could suggest that correctness is slightly negatively correlated with confidence.

Alternatively, it may also be that semi-formal models and metrics made participants less confident in their performance but still improved correctness. This could reflect on participants not trusting the models as much as directly inspecting the code and feeling less confident about results based on indirect information.

Additionally, the complexity of Kubernetes-based deployments may have influenced participant performance. Kubernetes manifests often require an understanding of multiple components, such as service discovery, networking, pod scheduling, and persistent storage management. Participants with limited experience in Kubernetes may have found it challenging to interpret these manifests, particularly in contrast to more straightforward Docker Compose configurations. The cognitive load associated with Kubernetes manifests may have resulted in increased task duration or a greater reliance on models and metrics to comprehend the system architecture.

This aligns with prior research suggesting that the complexity of infrastructure management tools varies significantly, and developers may struggle with tools that require greater abstraction reasoning. The study results indicate that while models improved correctness across all IaC types, understanding Kubernetes configurations remained challenging for participants compared to simpler declarative tools like Docker Compose. Future studies could investigate whether more extensive training in Kubernetes concepts or additional visualization tools could further aid comprehension.

Our results provide empirical evidence that formal models and metrics enhance correctness without increasing task duration. However, we do not claim that this effect is explained by specific cognitive mechanisms such as cognitive load reduction or mental model formation. Future studies could investigate the theoretical underpinnings of these findings in greater depth.

Further studies could also formally establish the negative relationship between models and confidence and explore its causes and effects. We proposed several possible factors, such as a lack of trust in the models or confidence negatively correlated with correctness, but these hypotheses should be formally tested. Furthermore, the implication of lowered confidence has not been explored.

7. Related Work

The following section will present a few works related to ours, focused on analyzing infrastructure as code.

Studies on infrastructure as code have been common since the industry has widely adopted this practice. Regarding analysis, the focus had mainly been on smells and practices. Sharma et al. [41] and Schwarz et al. [42] examine smells specifically for Puppet and Chef, respectively. Rahman et al. [6]'s work is another example of a work built around smells in IAC scripts. These approaches led to recommendations such as those in Hasan et al. [43], who identify six IaC testing practices to help practitioners improve the quality of infrastructure as code scripts. Similarly, the work of Kumara et al. [44] presents both best and bad practices based on an extensive grey literature review. Finally, Morris [3]'s book presents a collection of guidances on managing Infrastructure-as-Code. Our study differs from those in that it focuses on coupling as a quality measure, which is more complex to assess from code analysis than most smells.

Coupling is studied in Pendleton et al. [45] 's comprehensive metrics survey focused on cyber-attack defense interactions. Other works on metrics include Dalla Palma et al. [46, 47]'s catalog of 46 quality metrics for IaC scripts and Van der Bent et al. [48]'s metrics reflecting the best practices to assess Puppet code quality. As for tools, Wurster et al. [49] introduce TOSCA Lightning, an integrated toolchain for TOSCA Light, and Sotiropoulos et al. [50] present a tool for Puppet manifests and their system calls.

An analogous experiment to ours is that of Sandobalín et al. [51], who compare Argon with Ansible among 67 Computer Science students. Argon is a model-driven tool, and Ansible is a code-centric one, paralleling our own comparison of system analysis with or without the help of models. The study investigates effectiveness, efficiency, perceived ease of use, perceived usefulness, and intention to use. It concludes that Argon is more efficient, easier to use, and more helpful in specifying the infrastructure resources. A

key difference with our study is that it compares two different tools while our study compares the effectiveness of the same task with our without models.

Other works have studied the effect of models on the broader scope of software design in general: the experiment of Heijstek et al. [11] compares UML representations as diagrammatic artifacts and informal textual descriptions in communicating software design decisions to software developers based on a panel of forty-seven participants from industry and academia. The study finds no difference between the two mediums. A similar study by Labunets et al. [12] on security risk assessment on twenty-nine MSc students equally concludes that no significant differences exist between visual versus textual methods. Finally, Torchiano et al. [13] conducted a study on the comprehensibility of software design when using a class diagram combined with an object diagram compared to using a class diagram alone on a total of 124 participants. It also yielded mixed results, demonstrating no substantive time variation and a benefit in comprehensibility only for participants experienced with UML. In contrast, inexperienced participants seem to have performed worse when using two diagrams.

7.1. Threats to Validity

7.1.1. Threats to Internal Validity

The experimental sessions had no disruptions or incidents that interfered with the process. Participants received an introduction and were allowed to address any queries they had. No questions that broadly impacted the sessions emerged; instead, individual participant questions were addressed individually.

The limited time allocated to each session effectively minimized the potential for maturation effects; indeed, no such effects were observed. The experimental design ensured that each participant's contribution to the experiment results occurred only once, eliminating the possibility of learning between sessions. Therefore, any learning effect within a session across the tasks does not favor either *Control* or *Experimental*. Additionally, each participant had an equal opportunity to score points for their performance, irrespective of their assigned group, eliminating instrumental bias. Furthermore, the random assignment of participants to groups prevented any selection bias.

While preventing participants from discussing the experiment with future participants is impossible, measures were taken to minimize the potential for cross-contamination between experimental sessions and groups. Participants were not allowed to take a copy of the experiment sheet with them

or use electronic devices during the experiment. The complexity of the systems and tasks and the consecutive spacing of the sessions further reduced the likelihood of participants gaining unfair knowledge before their sessions. Additionally, randomly assigning participants to groups ensured that any advantage would be evenly distributed without favoring either group. The prohibition of electronic devices also prevented participants from accessing external information sources, as outlined in Section 4.2. The only permissible reference materials were the printed information document described in Section 3.5 and the limited access to the source code, thereby preventing potential effects of participants consulting other sources.

7.1.2. Threats to External Validity

A potential threat to external validity is the sample size of our study, which may be insufficient to yield statistically significant results. To mitigate this risk, we have employed robust statistical methods suitable for our sample size, ensuring the analysis is conducted appropriately and accurately given the available data.

7.1.3. Threats to Construct Validity

In Section 3.6, we examined *correctness* and *duration*, two commonly used dependent variables for assessing understandability. However, it is important to acknowledge that there may be alternative metrics that could better capture the concept of understandability.

While this study investigates coupling-related practices in IaC-based deployments, it primarily focuses on deployment strategies and infrastructure stack grouping. Data-sharing-related coupling (e.g., shared volumes in Kubernetes and Docker) and inter-networking coupling (e.g., service communication dependencies, networking policies, and API interactions) were not explicitly measured.

7.1.4. Threats to Content Validity

We cannot conceive of any threats to content validity since the experiment topic and subject matter were relevant to all participants' university courses, regardless of group assignment. Furthermore, the information sheet provided to participants provided sufficient prerequisite knowledge to participate in the study. Any inconsistencies or ambiguities in the experiment material would have affected both groups equally.

7.1.5. Threats to Conclusion Validity

Given that the experiment topic and subject matter were relevant to all participants' university courses, irrespective of their group assignment, we do not foresee any threats to content validity. Moreover, the information sheet provided to participants offered the necessary prerequisite knowledge for their participation in the study. Any inconsistencies or ambiguities in the experiment material would have affected both groups equally, ensuring fairness.

8. Conclusion

This study highlights the benefits of using semi-formal IaC system models and metrics to enhance practitioners' understanding of coupling-related practices in IaC-based deployments. Participants who had access to these models and metrics demonstrated greater accuracy than those who relied solely on source code, confirming that models are beneficial for improving comprehension.

It indicates that our semi-formal models and metrics could be helpful tools for a developer wishing to assess coupling-related practices on existing infrastructure, for an external reviewer or auditor unfamiliar with the application, or as an objective basis for discussion between different actors.

However, contrary to expectations, the use of models did not lead to a significant reduction in task *duration*. Although there was a time-saving effect for the SSA system, this trend did not hold across all tasks, particularly for the more complex ESC system. This suggests that while models help improve *correctness*, they do not always reduce *duration*, especially in more challenging scenarios.

Additionally, no significant relationship was found between task *duration* and *correctness*, except in the ESC *Control* group, where taking more time resulted in better performance. This indicates that spending additional time with the source code alone can improve understanding in simpler systems. At the same time, models offer the advantage of improving *correctness* without extending the time needed to complete tasks.

Interestingly, despite their better performance, participants who used models consistently reported lower confidence in their answers than those who did not. This suggests that although semi-formal models enhance *correctness*, they may also introduce a degree of uncertainty or cognitive load, making participants less confident in their responses. This finding aligns with

other studies that suggest a potential inverse relationship between confidence and *correctness* when using higher-level abstractions like models.

In conclusion, semi-formal IaC system models and metrics are effective tools for improving the understanding of coupling-related practices without negatively impacting task *duration*. However, the observed decrease in participant confidence points to the need for further investigation into how these models affect users' perceptions of their own performance. While we do not seek to validate cognitive theories, our findings suggest that formal models and metrics can serve as useful tools for improving understanding in software deployment contexts.

9. Acknowledgement

This research was funded in whole or in part by the Austrian Science Fund (FWF) project "Infrastructure-as-code Architecture Decision Compliance (IAC2)," project number: I 4731.

References

[1] J. Humble, D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley Professional, 2010.

[2] M. Nygard, Release It! Design and Deploy Production-Ready Software, Pragmatic Bookshelf, 2007.

[3] K. Morris, Infrastructure as Code: Dynamic Systems for the Cloud, Vol. 2, O'Reilly, 2020.

[4] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, D. A. Tamburri, Devops: Introducing infrastructure-as-code, in: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), 2017, pp. 497–498.

[5] L. Baresi, P. Spoletini, A. Susi, M. D. Penta, Towards the next generation of software engineering for smart contracts, ACM Transactions on Software Engineering and Methodology (TOSEM) 29 (2) (2020) 1–34.

[6] A. Rahman, C. Parnin, L. Williams, The seven sins: Security smells in infrastructure as code scripts, in: 2019 IEEE/ACM 41st International

Conference on Software Engineering (ICSE), 2019, pp. 164–175. doi:10.1109/ICSE.2019.00033.

[7] J. Humble, D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, Addison-Wesley Professional, 2010.

[8] Y. Brikman, Terraform: Up & running: Writing infrastructure as code (2020).

[9] A. W. Services, Aws well-architected framework: Operational excellence, https://aws.amazon.com/architecture/well-architected/ (2020).

[10] G. Cloud, Google cloud architecture framework: Best practices for cloud deployments, https://cloud.google.com/architecture/framework (2021).

[11] W. Heijstek, T. Kühne, M. R. V. Chaudron, Experimental analysis of textual and graphical representations for software architecture design, 2011 International Symposium on Empirical Software Engineering and Measurement (2011) 167–176.

[12] K. Labunets, F. Paci, F. Massacci, R. Ruprai, An experiment on comparing textual vs. visual industrial methods for security risk assessment, 2014. doi:10.1109/EmpiRE.2014.6890113.

[13] M. Torchiano, G. Scanniello, F. Ricca, G. Reggio, M. Leotta, Do uml object diagrams affect design comprehensibility? results from a family of four controlled experiments, Journal of Visual Languages & Computing 41 (2017) 10–21. doi:https://doi.org/10.1016/j.jvlc.2017.06.002.
URL https://www.sciencedirect.com/science/article/pii/S1045926X17301234

[14] P. Genfer, S. Serbout, G. Simhandl, U. Zdun, C. Pautasso, Understanding security tactics in microservice apis using annotated software architecture decomposition models ? a controlled experiment, Empirical Software Engineering: an international journal (February 2025). doi:10.1007/s10664-024-10601-1.
URL http://eprints.cs.univie.ac.at/8198/

[15] S. J. Warnett, U. Zdun, On the understandability of mlops system architectures, IEEE Transactions on Software Engineering (2024) 1–25doi:10.1109/TSE.2024.3367488.

[16] E. Ntentos, U. Zdun, J. Soldani, A. Brogi, Assessing architecture conformance to coupling-related infrastructure-as-code best practices: Metrics and case studies, in: 16th European Conference on Software Architecture, Software Architecture - 16th European Conference, ECSA 2022, Prague, Czech Republic, September 19?23, 2022, Proceedings, 2022, pp. 101–116. doi:https://doi.org/10.1007/978-3-031-16697-6_7.
URL http://eprints.cs.univie.ac.at/7338/

[17] C. Richardson, A pattern language for microservices, http://microservices.io/patterns/index.html (2017).

[18] U. Zdun, E. Navarro, F. Leymann, Ensuring and assessing architecture conformance to microservice decomposition patterns, in: M. Maximilien, A. Vallecillo, J. Wang, M. Oriol (Eds.), Service-Oriented Computing, Springer International Publishing, Cham, 2017, pp. 411–429.

[19] A. Jedlitschka, M. Ciolkowski, D. Pfahl, Reporting Experiments in Software Engineering, 2008, pp. 201–228. doi:10.1007/978-1-84800-044-5_8.

[20] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, Software Engineering, IEEE Transactions on 28 (2002) 721–734. doi:10.1109/TSE.2002.1027796.

[21] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer, Germany, 2012. doi:10.1007/978-3-642-29044-2.

[22] N. Juristo, A. Moreno, Basics of Software Engineering Experimentation, 2001. doi:10.1007/978-1-4757-3304-4.

[23] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, A. Pohthong, Robust statistical methods for empirical software engineering., Empirical software engineering. 22 (2) (2016) 579–630.
URL http://dro.dur.ac.uk/18658/

[24] G. Charness, U. Gneezy, M. A. Kuhn, Experimental methods: Between-subject and within-subject design, Journal of economic behavior & organization 81 (1) (2012) 1–8.

[25] M. Svahnberg, A. Aurum, C. Wohlin, Using students as subjects - an empirical evaluation, in: International Symposium on Empirical Software Engineering and Measurement, 2008.

[26] I. Salman, A. T. Misirli, N. Juristo, Are students representatives of professionals in software engineering experiments?, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1, 2015, pp. 666–676. doi:10.1109/ICSE.2015.82.

[27] S. S. Shapiro, M. B. Wilk, An analysis of variance test for normality (complete samples), Biometrika 52 (1965) 591–611.

[28] N. Mohd Razali, B. Yap, Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests, J. Stat. Model. Analytics 2 (01 2011).

[29] T. W. Anderson, D. A. Darling, A test of goodness of fit, Journal of the American Statistical Association 49 (268) (1954) 765–769. arXiv:https://www.tandfonline.com/doi/pdf/10.1080/01621459.1954.10501232, doi:10.1080/01621459.1954.10501232.
URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1954.10501232

[30] H. W. Lilliefors, On the Kolmogorov-Smirnov test for normality with mean and variance unknown, Journal of the American Statistical Association 62 (1967) 399–402.

[31] A. N. Kolmogorov, Sulla determinazione empirica di una legge didistribuzione, Giorn Dell'inst Ital Degli Att 4 (1933) 89–91.

[32] N. Cliff, Dominance statistics: Ordinal analyses to answer ordinal questions., Psychological Bulletin 114 (1993) 494–509.

[33] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, A. Pohthong, Robust statistical methods for empirical software engineering, Empirical Softw. Engg. 22 (2) (2017)

579–630. doi:10.1007/s10664-016-9437-5.
URL https://doi.org/10.1007/s10664-016-9437-5

[34] N. Cliff, Answering ordinal questions with ordinal data using ordinal statistics, Multivariate Behavioral Research 31 (2010) 331–350. doi:10.1207/s15327906mbr3103_4.

[35] H. D. Delaney, A. Vargha, Comparing several robust tests of stochastic equality with ordinally scaled variables and small to moderate sized samples., Psychological methods 7 4 (2002) 485–503.

[36] L. M. Hsu, Biases of success rate differences shown in binomial effect size displays., Psychological methods 9 2 (2004) 183–97.

[37] Y. Benjamini, Y. Hochberg, Controlling the false discovery rate - a practical and powerful approach to multiple testing, J. Royal Statist. Soc., Series B 57 (1995) 289 – 300. doi:10.2307/2346101.

[38] C. Bonferroni, Teoria statistica delle classi e calcolo delle probabilita, Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze 8 (1936) 3–62.

[39] O. J. Dunn, Multiple comparisons among means, Journal of the American Statistical Association 56 (1961) 52–64.

[40] J. H. Bray, S. E. Maxwell, Analyzing and interpreting significant MANOVAs, Review of Educational Research 52 (3) (1982) 340–367.
URL http://www.jstor.org/stable/1170422

[41] T. Sharma, M. Fragkoulis, D. Spinellis, Does your configuration code smell?, in: Proceedings of the 13th International Conference on Mining Software Repositories, MSR '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 189–200.

[42] J. Schwarz, A. Steffens, H. Lichter, Code smells in infrastructure as code, in: 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC), 2018, pp. 220–228. doi:10.1109/QUATIC.2018.00040.

[43] M. Mehedi Hasan, F. A. Bhuiyan, A. Rahman, Testing practices for infrastructure as code, 2020, pp. 7–12. doi:10.1145/3416504.3424334.

[44] I. Kumara, M. Garriga, A. U. Romeu, D. Di Nucci, F. Palomba, D. A. Tamburri, W.-J. van den Heuvel, The do's and don'ts of infrastructure code: A systematic gray literature review, Information and Software Technology 137 (2021) 106593.

[45] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, S. Xu, A survey on systems security metrics, ACM Comput. Surv. 49 (4) (Dec. 2016).
URL https://doi.org/10.1145/3005714

[46] S. Dalla Palma, D. Di Nucci, F. Palomba, D. A. Tamburri, Toward a catalog of software quality metrics for infrastructure code, Journal of Systems and Software 170 (2020) 110726.

[47] S. Dalla Palma, D. Di Nucci, D. A. Tamburri, Ansiblemetrics: A python library for measuring infrastructure-as-code blueprints in ansible, SoftwareX 12 (2020) 100633.

[48] E. van der Bent, J. Hage, J. Visser, G. Gousios, How good is your puppet? an empirically defined and validated quality model for puppet, in: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2018, pp. 164–174. doi:10.1109/SANER.2018.8330206.

[49] M. Wurster, U. Breitenbücher, L. Harzenetter, F. Leymann, J. Soldani, Tosca lightning: An integrated toolchain for transforming tosca light into production-ready deployment technologies, in: N. Herbaut, M. La Rosa (Eds.), Advanced Information Systems Engineering, Springer International Publishing, Cham, 2020, pp. 138–146.

[50] T. Sotiropoulos, D. Mitropoulos, D. Spinellis, Practical fault detection in puppet programs, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 26–37.

[51] J. Sandobalín, E. Insfran, S. Abrahão, On the effectiveness of tools to support infrastructure as code: Model-driven versus code-centric, IEEE Access (01 2020). doi:10.1109/ACCESS.2020.2966597.